x86 AVX 命令を用いた多倍長精度線型計算ライブラリ BNCmatmul の性能評価

Performance Evaluation of BNCmatmul, multiple precision linear computation library, accelerated with x86 AVX instruction

幸谷智紀*

Tomonori KOUYA

Abstract: We are currently trying to accelerate multiple precision basic linear computation on several x86_64 CPUs. Our aim is mainly for turning those based on multi-component type floating-point arithmetic, which can be implemented with some components of IEEE754 binary64 and error free transformation technique. There are vast kinds of basic linear computation to be confirmed via benchmark tests. Although we cannot unveil all of them, we describe the facts about some issues until now.

1. 初めに

現在の科学技術計算において，有限桁の浮動小数点演算が不可欠であることは論を俟たないが，どの程度の長さの仮数部が必要になるかは問題の性質，使用する数値計算アルゴリズム，そしてユーザの要求次第である．ことに，数値的に安定な線型計算を多数含む深層学習の分野では，仮数部の桁数を減らしてでも高速性を要求する傾向が強い．それ故に結果の再生性 (reproductibity) が問題になることがある．膨大な計算を行った結果，IEEE754 単精度 (binary32, 仮数部 24bits) より小さい半精度 (bianry16) では学習結果の再生産性が得られず，それでも統計的な学習結果にはさほど依存しないということで敢えてその問題には向き合わないということが多いように見受けられる．

とはいえ，binary16 から MPFR[3] が提供する任意精度まで，問題特性とユーザのニーズに応じて可変的に仮数部長だけでなく指数部・符号部も含む浮動小数点数フォーマットを選択する必要があることは万人が認めるところであろう．Bailey[1] は「可変精度計算 (variable precision)」の時代であることを述べ，現在に必要とされる下記の 10 項目の課題を取り上げている．

1. 純粋数学から応用志向の科学技術まで含めて，どのように可変精度使用の原則を確立すればよいか？
2. 爆発的に成長し続けている人工知能・機械学習に対してどんなハードウェア・ソフトウェアの道具立てを用意して可変精度をサポートするのか？　これらに対して基本的な最終結果の精度要求はどのようなものになるのか？
3. 2〜3 桁の精度で十分という計算から，数百・数千桁が必要となる計算まで，高速に実行できてかつ正確な演算とライブラリ関数を求める新たな数学的なアプローチ方法はあるのか？
4. 例えば　Gustafson and Yonemoto (2017); Lindstrom et al. (2018) らが提唱するような，可変精度計算のための新たなシステムや数の表現方法はあるのか？　既存のハードウェアベースの IEEE754 演算や MPFR が持つ強味と弱味を克服できる新たな提案はあるのか？
5. ハードウェア・ソフトウェアベンダーから必要とされる可変精度計算に対するサポート要求はないのか？ 全てソフトウェアベースの実装に頼るのではなくて，ハードウェアの多少の変更程度で低精度，可変精度は実現できたりしないのか？
6. 殆どの数値計算・数値解析関連の研究論文は 32bit もしくは 64bit の計算結果しか扱っていない．可変精度計算環境であればもっと良い成果が得られるという技法は無いものか？
7. Wilkinson が提唱した，数値計算における古典的な誤差限界は最悪な場合のみ扱っており，精度が限られていたり，可変精度の環境ではあまり有効ではない．Higham and Mary (2018) が提唱したような，もっと現実的な誤差の見積もりはできないものか？
8. 可変精度計算環境で，どうやったら反復改良法のベストな方法で動作させられるか？
9. 浮動小数点数を素早く圧縮してストレージに格納したり転送したりするアルゴリズムやソフトウェア的方法はあるか？
10. 自動的もしくは半自動的にアプリケーションコードを解析し，数値的に敏感なところをを探し出して手当の方法を提案してくれるようなソフトウェアツールはないか？

我々は現在広く使用されている多倍長精度ライブラリである QD および MPFR をベースに，基本線型計算から非線形問題まで幅広く多倍長精度を必要とする問題に対して実装を伴った現実的な改良方法を提唱してきた．ここ数年来は特にマルチコンポーネント方式の基本線型計算の高速化に従事しており，2020 年から科研費を取得して主として x86 CPU に備わる SIMD 命令である AVX2 を活用した高速化を行ってきた．2021 年からは新たに AVX-512F が使え

2022 年 5 月 27 日 受理

* 情報学部 コンピュータシステム学科

る Xeon マシンも購入し，ベンチマークテストを行ってきたが，はかばかしい高速化が達成できず, Intel からはコンシューマ向けの CPU では AVX-512 命令群を使用不可とする方向であるとのアナウンスまで行われる始末である．

本稿では今までの我々のマルチコンポーネント方式の多倍長精度線型計算の高速化に関する研究をまとめて紹介し，現時点で判明しているベンチマーク結果を述べる．基本線型計算に限ってのベンチマーク項目が膨大になるため，全てを網羅することは出来ていないが，AVX-512F の利用によってどのような変化が，特に AVX2 との比較において起きるのか，コンパイラによる計算時間の違いなど，現在我々が使用できる環境下でベンチマークした結果に基づいて報告する．

2. 多倍長精度浮動小数点演算の方式と代表的なライブラリ

現状，ハードウェアの動作周波数を上げることは，消費電力や発熱の問題から難しく，性能向上はもっぱら下記の２つの方法で行われるようになっている．

- SIMD 命令の強化，コア数の増加といった並列処理能力の向上
- 半精度・単精度浮動小数点演算の導入・強化といった，処理のニーズに応じた浮動小数点数の仮数部長の増減

ソフトウェア的には，これらのハードウェアの性能向上策を利用し，SIMD 命令ライブラリ，OpenMP，MPI といった並列技法を駆使しつつ，ハードウェアが標準的に備える IEEE 単精度・倍精度 (Binary32, Binary64) と半精度を組み合わせた混合精度技法を用いた性能向上策が取られるようになっている．

しかしながら，科学技術計算そのもののニーズは衰えるところを知らず，binary64 では計算精度が不足する悪条件問題はさまざまなジャンルで散見される．そのような悪条件問題は仮数部の桁数を増やした多倍長精度浮動小数点演算（多倍長計算）によって解決できることが多く，Bailey が主張するように，ユーザの要求精度を満足する多倍長精度ソフトウェア，もしくはライブラリのニーズも高まる一方である．

現状の多倍長精度浮動小数点演算は，無誤差変換技法を用いて binary32, 64 を複数組み合わせて仮数部長を伸ばすマルチコンポーネント方式と，整数ベースの多数桁方式の 2 種類の実装方式に基づくものが主流である．マルチコンポーネント方式は Bailey らの QD ライブラリが著名であり，多数桁方式では GNU MP(GMP) の任意長自然数カーネル (MPN) を用いた MPFR ライブラリが高速性と信頼性の両方に優れたものとして多数のユーザを抱えている．中田による MPLAPACK はこの 2 つの多倍長精度浮動小数点演算ライブラリの土台の上に開発された LAPACK/BLAS の Fortran コードに基づく多倍長精度線型計算ライブラリであり，2022 年 3 月現在の最新バージョンは Version 1.0.1 で，Version 2.0 は鋭意開発中のようである．OpenMP による BLAS コードの並列化に対応し，LAPACK の主要なドライバルーチンや計算ルーチンを多倍長精度で利用できるようになっている．しかしながら，全てのドライバルーチンがこの高性能 MPBLAS の恩恵を預かるにはまだ時間がかかると思われる．また SIMD 命令，CUDA の利用，尾崎スキームの導入による高性能化は行われていない．

そこで，多倍長精度線型計算ライブラリにもソフトウェア的性能向上策を積極的に取り入れて高速化する研究は以前から行われており，Table 1 に示すように，各精度計算・環境において有望な最適化技法は研究され尽くされつつある．

Table 1　多倍長精度行列乗算の研究・ライブラリ一覧 (2022 年 3 月現在)

xGEMM	CPU				GPU	
Opt.Method	None	AVX2	OpenMP	Ozaki Scheme	CUDA	Ozaki Scheme
DS	?	?	?	?	Mukunoki	Mukunoki, Nanai
TS	?	?	?	△ Utsugiri	○ Utsugiri	△ Utsugiri
QS	?	?	?	Utsugiri	Utsugiri	Utsugiri
IEEE754 Binary128	MPBLAS	?	MPBLAS	Mukunoki	Mukunoki	Mukunoki
DD	MPBLAS, BNCmatmul	Lis, MuPAT BNCmatmul	Lis,MuPAT, MPBLAS, BNCmatmul	?	Mukunoki	Mukunoki
TD	BNCmatmul	BNCmatmul	BNCmatmul	○ Utsugiri	△ Utsugiri	△ Utsugiri
QD	MuPAT, MPBLAS, BNCmatmul	BNCmatmul	MuPAT, MPBLAS, BNCmatmul	Utsugiri	Utsugiri	Utsugiri
MPFR	MPBLAS, BNCmatmul	?	MPBLAS, BNCmatmul	?	CUMP	?

MPLAPACK/MPBLAS	https://github.com/mahonakata/mplapack
MuPAT	Yagi, H et.al(2020)
Lis	https://www.ssisc.org/lis/
CUMP	https://github.com/skystar0227/CUMP
BNCmatmul	https://na-inet.jp/na/bnc/
Mukunoki	Mukunoki, D et.al.(2021)
Utugiri	打桐(2021)

我々の BNCmatmul ライブラリ [5] はもっぱらマルチコア CPU 上での最適化を行ったものだが，尾崎スキームを用いた高速化にも取り組んでいる．以下，BNCmatmul のソフトウェア構成と最適化技法について簡単に述べる．

3. BNCmatmul の構成

我々の BNCmatmul は，前身となる BNCpack[2] から派生してできたもので，ANSI C を用いて MPFR による任意精度線型計算を高速化するために Strassen，Winograd アルゴリズムを取り入れて分離して単独のライブラリとしたものである．その後，マルチコンポーネント方式の DD(Double-double, 仮数部長 106bits), TD(Triple-double, 159bits), QD(Quadruple-double, 212bits) を，C++ のクラスライブラリではなく，ANSI C の範囲内でインライン関数として実装したものを取り込み，後述するように AVX2, AVX-512F の SIMD 命令を C 関数で組み込んで最適化を行った．そのソフトウェア構成図を Fig.1 に示す．

現状では x86_64 系統の CPU 向け最適化しか行っていないが，Arm 等の有力な CPU ファミリーが台頭してくれば，SIMD 命令を用いた最適化を行っていきたいと考えている．

3.1 マルチコンポーネント方式による DD, TD, QD 精度基本線型計算

ここではマルチコンポーネント方式の基本線型計算に対する最適化技法を述べる．基本，小武守ら [4] が行った一連の SIMD 命令使用による最適化技法を，DD(Double-double, 仮数部 106bits-全長 128bits) だけでなく TD(Triple-double,

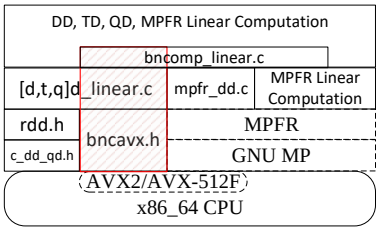

Fig.1　BNCmatmul のソフトウェア構成

159bits-192bits), QD(Quadruple-double, 212bits-256bits) にも適用したものである．我々は更にデータ構造も AVX2 (256bits ベクトルが基本), AVX-512F(512bits ベクトル) に最適化し，各コンポーネントごとに binary64 一次元配列としてベクトル，行列を定義し，読み出し・書き込みをそれぞれ 256, 512bits ごとに実行できるようにしている．TD ベクトル演算を例に AVX2, AVX-512F でどのような処理を行っているかを Fig.2 に示す．

この方式に従ってどの程度性能向上が図れるかを実際にベンチマークテストを行って計測し，DD, TD, QD それぞれで基本線型計算全般が高速化できることを確認している．

3.2　MPFR ベースの任意精度基本線型計算

MPFR の源流である GNU MP(GMP) は ANSI C で作られた，任意精度自然数演算ライブラリ (MPN) を核とする C 関数群であり，MPFR も MPN カーネルを土台に ANSI C で構築された任意精度演算ライブラリである．従って，GMP や MPFR を土台に C++ のクラスライブラリとして構築されたものはどうしても処理にオーバーヘッドが入り込み，直接 GMP や MPFR の C 関数を呼び出すより時間がかかる傾向がある．

MPBLAS が任意精度計算で用いている MPFR のラッパー C++ クラスライブラリである mpreal も例外ではなく，MPBLAS の任意精度ルーチン群は，我々が実装した C ネイティブの基本線型計算より低速になることが判明している．実際，下記の EPYC，Xeon 両環境でベンチマークテストを実行した結果を示す．

EPYC　AMD EPYC 7402P 24 cores, Ubuntu 18.04.6 LTS, GCC 7.5.0, Intel Compiler version 2021.4.0, MPLAPACK 1.0.1, BNCpack 0.8, MPFR 4.1.0

Xeon　Intel Xeon W-2295 3.0GHz 18 cores, Ubuntu 20.04.3 LTS, GCC 9.3.0, Intel Compiler version 2021.5.0, MPLAPACK 1.0.1, BNCpack 0.8, MPFR 4.1.0

我々の BNCmatmul ライブラリがサポートする MPFR ブロック化行列乗算の計算時間を，MBLAS の MPFR Rgeem の時間と比較したものを Fig.3 に示す．比較のために Strassen 行列乗算の結果も載せている．

これでわかるように，EPYC，Xeon どちらの環境でも，212bits，1024bits 共に，MBLAS(Rgemm) に対し，ブロック化行列乗算 (matmul_mpfmatrix_block) で 2.0〜2.4 倍 (212bits), 1.4〜1.5 倍 (1024bits) の差が出る．従って，計算桁数が小さいほど，MPLAPACK/MPBLAS(mpreal) より C ネイティブの MPFR ダイレクト呼び出し (BNCmatmul) が高速になる事がわかる．また，Strassen 行列乗算は，行列サイズや精度が大きくなるにつれて性能向上率がアップしていることもわかる．

我々の BNCmatmul における任意精度線型計算は，OpenMP による並列化を行うのみで，線型計算に対する SIMD 命令を直接用いた高速化は行っていない．それでも MPFR 関数群を直接呼び出して使用していることで，これだけの高速化が図れていることになる．

4.　いくつかの性能評価

本節では特に AVX-512F を用いたベンチマークテスト結果を中心に述べる．AVX2 を用いたマルチコンポーネント方式行列乗算の高速化についてはすでに示しているので，詳細はそちらに譲り，AVX-512F の結果との比較多少としてのみ使用する．

4.1　Intel コンパイラと Gnu Compiler Collection 使用時の比較

Intel コンパイラ (ICC) は x86_64 環境下では，特に最適化性能については定評にある商用コンパイラで，２０２２年３月現在はアカデミック用途で無料で使用できるものが提供されている．GNU Compiler Collection(GCC) は標準的な Linux ディストリビューションでは必ず使えるものである．

ここでは ICC, GCC による AVX2, AVX-512F を用いたベクトル要素同士の加算と減算ベンチマークテスト結果を Fig.4 に示す．

これを見る限り，ICC より GCC で AVX-512F の性能向上が図れている．しかしながら，実際の線型計算，行列乗算では少し奇妙な結果をもたらしている．

4.2　AVX-512F 使用時の性能評価

前述したように，現状，少なくともコンシューマ向けの Intel CPU において，AVX-512F はあまり性能が芳しくない．整数演算では効能があるという報告もあるにはあるが，少なくとも我々が Intel Compiler や GCC を用いてベンチマークテストを行った範囲では，AVX2 より全般的な性能向上が図れるという結果は得られていない．しかしながら，シリアル計算でのみ，AVX-512F による（かもしれない）性能向上例が得られたのでここに報告しておく．

なお，ここで新たに AVX-512F が使用可能である Corei9 環境をベンチマークテスト環境として追加しておく．

Corei9　Intel i9-10900X 3.7GHz 10 cores, Ubuntu 20.04.2 LTS, GCC 9.4.0, Intel Compiler version 2021.3.0, MPLAPACK 1.0.1, BNCpack 0.8, MPFR 4.1.0

Corei9 の結果を Fig.5 に示す．

明らかに，シリアル計算時の AVX-512F 利用時の計算性能が 8〜12 倍向上していることがわかる．反面，OpenMP 並列実行時においては AVX2 との性能向上は全く見られない．

現時点ではこの理由は不明であるが，もともと CPU 内の演算リソースが足りない時には AVX の性能が出ないということが経験的に知られているようで，コンシューマ向けの CPU においては AVX-512F の性能向上はせいぜいシリアル

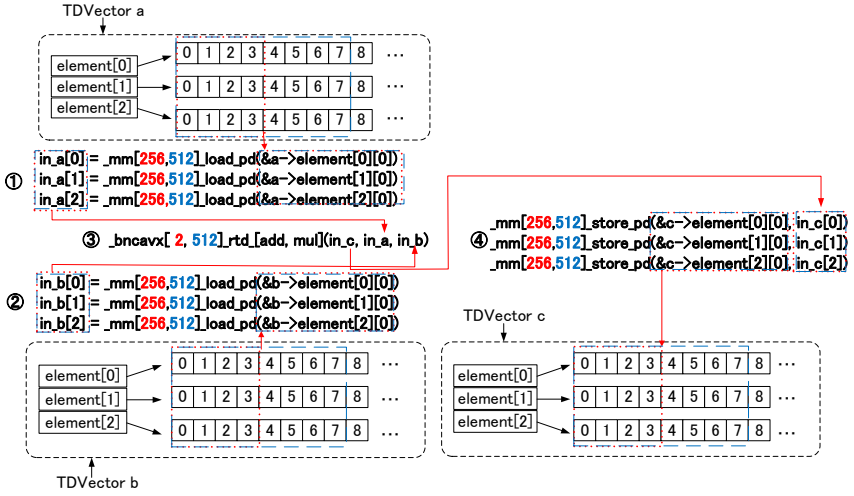

Fig.2　AVX2 および AVX-512F による TD 精度ベクトル演算のアクセス方式

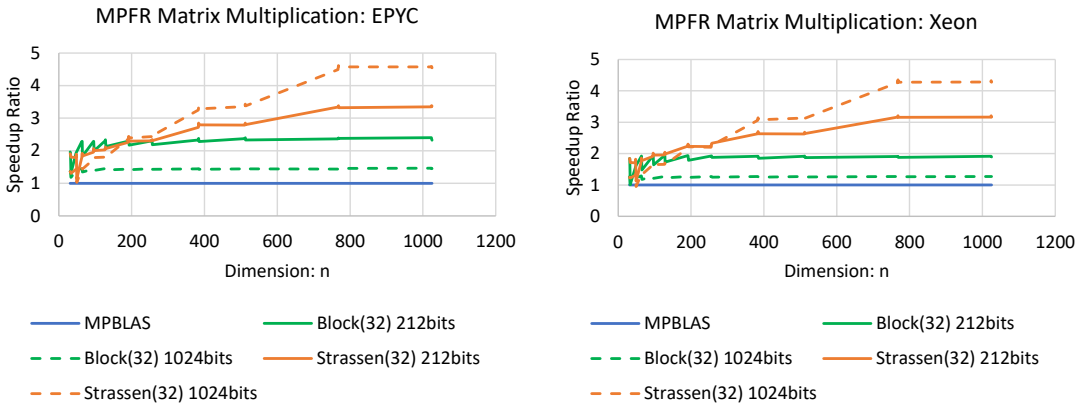

Fig.3　MPBLAS(Rgemm) に対する MPFR 行列乗算の高速化率：EPYC(左) と Xeon(右)

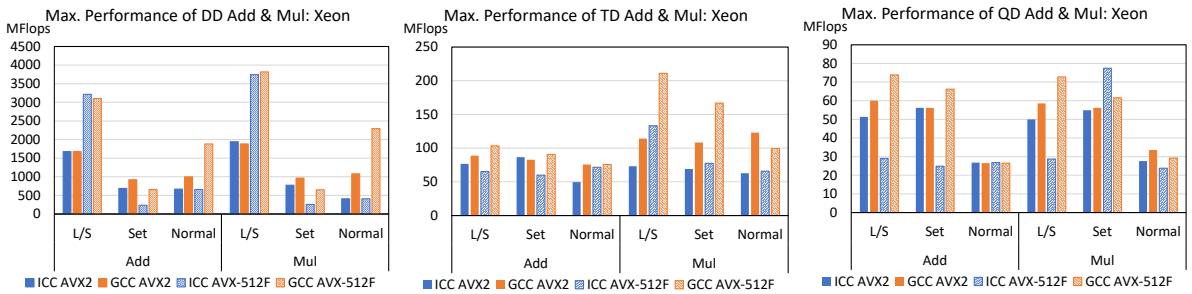

Fig.4　ベクトル演算 (加算と乗算) の最大性能: Xeon

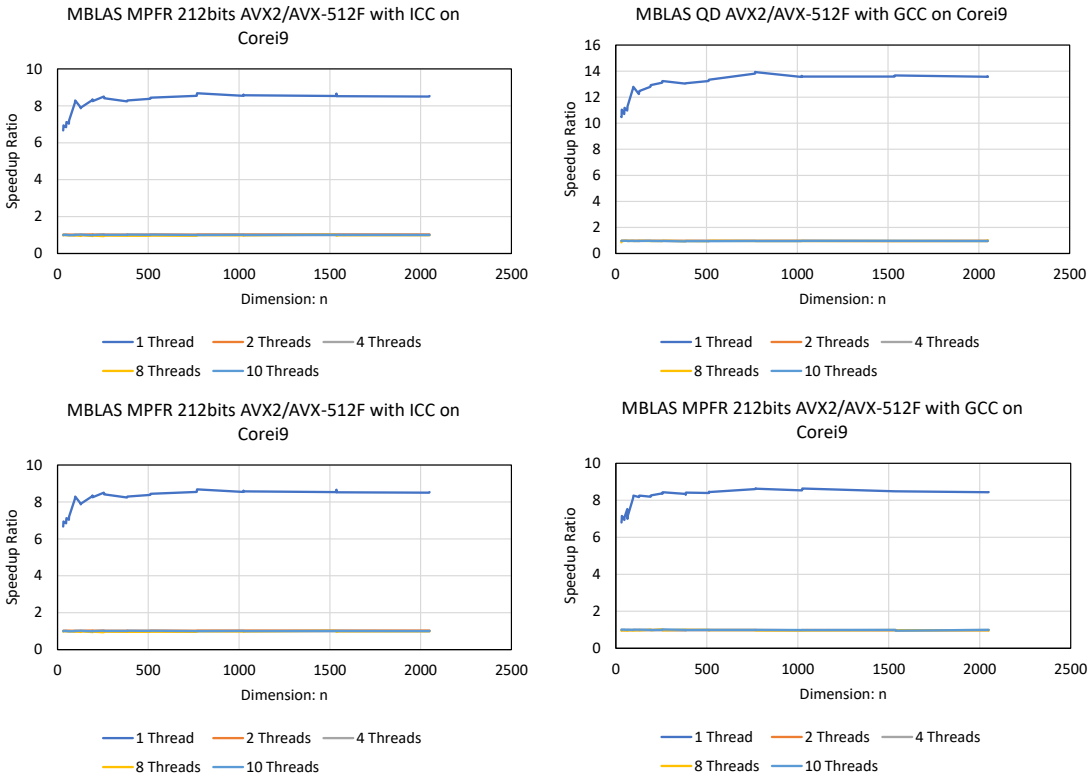

Fig.5　AVX-512F 利用による MPBLAS Rgemm の性能向上比率: Corei9

計算止まり，ということなのかもしれない．いずれにせよ，今後 Intel は一般向けの AVX-512F は obsolete 扱いにするようで，一般利用を考えると，AVX-512F 最適化の機能はオプション扱いにしておくことが無難と考える．

5. まとめと今後の展開

以上，現状の多倍長精度線型計算の最適化結果を示した．今後は他の基本線型計算，特に行列・ベクトル演算 (MV) を中心にベンチマークテスト結果をまとめ，Python ライブラリを含む BNCmatmul をリリースしていきたい．

謝辞

本研究の一部は科学技術研究費 20K11843 の助成を利用して行われた．また静岡理工科大学研究支援費の支援も受けている．関係各位に感謝する．また，丁寧な査読を行って頂いた未知の査読者にも感謝する．

参考文献

1) David H Bailey. Reproducibility and variable precision computing. *The International Journal of High Performance Computing Applications*, Vol. 34, No. 5, pp. 483–490, 2020.
2) Tomonori Kouya. BNCpack. `https://na-inet.jp/na/bnc/`.
3) MPFR Project. The MPFR library. `https://www.mpfr.org/`.
4) T.Kotakemori, S. Fujii, H. Hasegawa, and A. Nishida. Lis: Library of iterative solvers for linear systems. `https://www.ssisc.org/lis/`.
5) T.Kouya. BNCmatmul. `http://na-inet.jp/na/bnc/bncmatmul-0.2.tar.bz2`.